



HEREBY CERTIFY THAT THIS CORRESPONDENCE IS
BEING DEPOSITED WITH THE U.S. POSTAL SERVICE
AS FIRST CLASS MAIL ADDRESSED TO:
U.S. PATENT AND TRADEMARK OFFICE,
P.O. BOX 2327, ARLINGTON, VA 22202, ON
DATE OF DEPOSIT: FEBRUARY 28, 2003
PERSON MAKING DEPOSIT: ANNE VACHON DOUGHERTY

Anne Vachon Dougherty 2/28/03
Signature & Date

RECEIVED
MAR 11 2003
Technology Center 2100

#13
3/13/03
A.W.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

<u>In Re Application of</u>	February 27, 2003
<u>H.W. ADAMS, ET AL</u>	Group Art No.: 2175
<u>Serial No.: 09/421,139</u>	Examiner: S. Rimell
<u>Filed: October 19, 1999</u>	Anne Vachon Dougherty 3173 Cedar Road Yorktown Hts, NY 10598
<u>Title: SYSTEM AND METHOD FOR INTERACTIVE READING AND LANGUAGE INSTRUCTION</u>	

REQUEST FOR CONSIDERATION

Commissioner for Patents

Sir:

In response to the Office Communication dated January 27, 2003, Applicants submit the following response to the two matters raised by the Examiner:

- 1) Applicants filed a submission entitled "Amendment" on November 18, 2002.

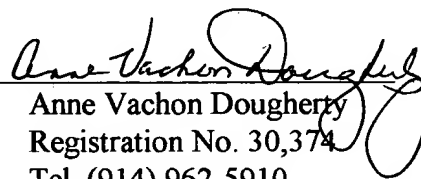
The Examiner has noted that the submission contains no amendments to either the Specification or the Claims and requests clarification. Applicants acknowledge that the

submission contains no changes to the Specification or the Claims. Applicants nonetheless respectfully request consideration of the arguments submitted therein in support of the patentability of the subject invention.

2) The documents in support of the Declaration of Prior Invention are enclosed herewith. The Declaration establishes that the claimed invention was invented prior to the effective date of the Mostow patent. While Applicants are able to swear behind the Mostow reference, Applicants believe that the Mostow patent does not anticipate the invention as claimed, as argued in the response filed on November 18, 2002.

In light of the foregoing, it is respectfully requested that the Response(s) be entered, the rejections be reconsidered and withdrawn, and the remaining Claims 1-9, 11-14, 16-19, 21-23, 31-36, 38-39 and 44-51 be passed to issuance.

Respectfully submitted,
H. W. Adams, et al

By: 
Anne Vachon Dougherty
Registration No. 30,374
Tel. (914) 962-5910

Enclosures



John Dougherty Fax 962-1973

RECEIVED

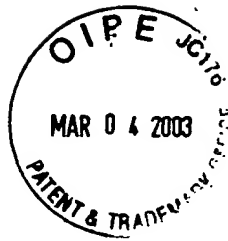
MAR 11 2003

Technology Center 2100

Here is a screen dump of the archive made on 01/03/96. It references files via their last modification date, unfortunately, but you can see that there are files unmodified since 06/12/95.

On subsequent pages I have included the contents of some of the files.

File	Date	Time	Size	Perm	Owner
back.bat	11/07/95	12:52	128	36%	82
get.bat	11/07/95	12:52	46	33%	31
list.bat	11/18/95	22:01	33	0%	33
make.bat	11/20/95	12:16	22	0%	22
update.bat	11/07/95	15:54	36	0%	36
bgtbl.c	11/18/95	22:30	2,228	84%	358
book.c	01/03/96	12:05	4,962	84%	787
color.c	10/13/95	14:53	369	51%	180
file.c	11/20/95	15:44	1,035	49%	530
getid.c	01/03/96	12:01	6,999	73%	1,877
global.c	11/20/95	14:38	4,577	66%	1,565
init.c	11/20/95	14:59	3,829	69%	1,179
page1.c	09/18/95	15:01	914	66%	310
paint.c	11/20/95	14:36	8,608	80%	1,757
parse.c	01/03/96	11:56	2,686	64%	972
perform.c	11/08/95	17:22	6,269	73%	1,692
read.c	11/20/95	15:55	14,317	74%	3,789
res.c	11/07/95	12:50	1,453	63%	542
rgb.c	11/07/95	12:48	2,198	62%	825
sndout.c	11/20/95	14:40	2,579	61%	1,013
tables.c	11/20/95	12:16	5,602	87%	741
text.c	11/20/95	15:29	23,901	81%	4,604
vt.c	11/20/95	15:29	9,180	74%	2,404
vtmsg.c	11/14/95	16:41	11,676	79%	2,410
dialogs.dlg	11/18/95	22:45	1,619	65%	565
dialogs.h	11/18/95	20:43	520	76%	127
global.h	11/20/95	14:38	5,369	70%	1,604
read.h	01/03/96	11:42	7,321	68%	2,314
resource.h	11/18/95	19:04	489	48%	254
struct.h	11/07/95	12:51	2,060	66%	697
messy.ico	10/25/95	22:19	766	69%	238
reader.ico	06/12/95	15:51	766	73%	208
read.rc	01/03/96	11:59	6,422	74%	1,658



RECEIVED

MAR 11 2003

Technology Center 2100

This is the file READ.C

```
#include "read.h"
#include "global.h"
#include "dialogs.h"
#include <windows.h>
#include <smapi.h>
#include <mmsystem.h>
```

```
int APIENTRY WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszC-
mdLine,
```

```
    int nCmdShow)
```

```
{
```

```
    /*
```

```
    * local storage
```

```
    */
```

```
    HWND  hWnd;
```

```
    MSG   msg;
```

```
    /*
```

```
    * routines called
```

```
    */
```

```
    LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
```

```
    void  initRes();    /* init resolution values */
```

```
    void  initMem();    /* init memory */
```

```
    if (!hPrevInstance){
```

```
        /*
```

```
        * register the main window class
```

```
        */
```

```
        wndclass.style      = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;
```

```
        wndclass.lpfnWndProc = WndProc;
```

```
        wndclass.cbClsExtra  = 0;
```

```
        wndclass.cbWndExtra  = 0;
```

```
        wndclass.hInstance  = hInstance;
```

```
        wndclass.hIcon      = LoadIcon(hInstance, "GAME_ICO");
```

```
        wndclass.hCursor    = NULL;
```

```
        wndclass.hbrBackground = GetStockObject(BLACK_BRUSH);
```

```
        wndclass.lpszMenuName = NULL;
```

```
        wndclass.lpszClassName = "MainClass";
```

```
        if (!RegisterClass (&wndclass))
```

```
            return(0);
```

```
    }
```

```

/*
 * get path to user data directory
 */
GetPrivateProfileString("Paths", "Users", "", userPath,
                        SIZ_PATH_BUF, "watchme.ini");

GetPrivateProfileString("Paths", "Wave", "", wavPath,
                        SIZ_PATH_BUF, "watchme.ini");

/*
 * get the number of attempts to make before advancing to the next
 * word
 */
GetPrivateProfileString("Parms", "Attempts", "5", cBuf,
                        SIZ_ERR_BUF, "watchme.ini");
attempts = atoi(cBuf);

/*
 * get whether to provide feedback
 */
GetPrivateProfileString("Parms", "Feedback", "TRUE", cBuf,
                        SIZ_ERR_BUF, "watchme.ini");
feedBack = strcmp("TRUE", cBuf) ? FALSE : TRUE;

/*
 * get the minimum score on words for a match
 */
GetPrivateProfileString("Parms", "Score", "10", cBuf,
                        SIZ_ERR_BUF, "watchme.ini");
minScore = atoi(cBuf);

/*
 * get the rejection threshold
 */
GetPrivateProfileString("Parms", "Reject", "5", cBuf,
                        SIZ_ERR_BUF, "watchme.ini");
rejectVal = atoi(cBuf);

initRes();      /* initialize the resolution values */
vtdStat = 0;    /* flag VTD is uninitialized */
initFlg = FALSE; /* flag user not initialized */
advFlg = FALSE; /* advance word active flag */
appState = STAT_READ; /* init state to reading */

```

```

hFile = INVALID_HANDLE_VALUE; /* flag no data file open */
initMem();    /* initialize memory */

/*
 * load the frequently used strings
 */
LoadString(hInstance, S_APPNAME, appName, SIZ_STR_BUF);

/*
 * create the program window
 */
hInst = hInstance;
hWnd = CreateWindow ("MainClass", appName, WS_POPUP,
    0, 0, width, height+capY, NULL, NULL, hInstance, NULL);
hCursor = LoadCursor(NULL, IDC_ARROW);
SetCursor(hCursor);
ShowWindow (hWnd, nCmdShow); /* display the window */

while(GetMessage(&msg, (HWND)NULL, (UINT)NULL, (UINT)NULL)){
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
return(msg.wParam);
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT wMessage, WPARAM wParam,
LPARAM lParam)
{
    /*
     * local storage
     */
    PAINTSTRUCT ps;
    LONG    wID;    /* control ID */
    int     rc;     /* return code */
    SM_MSG  reply;  /* VTD reply message */
    char    msgBuf[SIZ_ERR_BUF]; /* error string message buffer */
    HBRUSH  hTempBrush; /* temp brush used to free custom brushes */
    DWORD   errCode; /* file system error code */
    int     nTimer; /* # of timer for this application */

    char    *kidImage[] = {
        "asian girl",
        "black girl",
        "hispanic girl",
        "white girl",
    }

```

```

    "asian boy",
    "black boy",
    "hispanic boy",
    "white boy",
};

```

```

static int expLeft;
static int expRight;
static int expTop;
static int expBot;

```

```

/*
 * routines called
 */
void initGraph(HWND);      /* init graphics objects */
int vtdOpen(HWND);         /* open the VTD session */
void vtdErr(HWND, int, int); /* display VTD error */
void paint(HWND);          /* repaint the window */
void vtdMsg(HWND, UINT, WPARAM, LPARAM); /* process VTD messages */
BOOL CALLBACK getID(HWND, UINT, WPARAM, LPARAM); /* get user ID dialog
proc */
void fileErr(int, DWORD);  /* report file error */
void advWord(HWND);        /* advance to the next word */
void caution();            /* display the word in caution text color */
void selWord(HWND, WORD, WORD); /* try to select a word */
void advFin(HWND);         /* finish the advance to the next word */
void sndOut(HWND);         /* play a sound */
void playNext(HWND);       /* play back the next set of words */
void raise(HWND);          /* raise the curtain */
BOOL tstNext(HWND, WORD, WORD); /* test if next page button hit */
BOOL tstPrev(HWND, WORD, WORD); /* test if previous page button hit */
int micOn(HWND);           /* turn on the microphone */
BOOL wrtLog(HWND, char *); /* write to the log file */
void perform(HWND); /* begin reading the performance */

```

```

wID = LOWORD(wParm);

```

```

switch (wMessage)
{
    case WM_TIMER:
        if (appState == STAT_CURTAIN)
            raise(hWnd);
        break;

```

```

case WM_COMMAND:
    switch( wID ) {
        case IDM_CONNECT_ID:
            vtdMsg(hWnd, wMessage, wParam, lParam);
            break;
    }
    break;

case WM_CLOSE:
    DestroyWindow(hWnd);
    break;

case WM_CREATE:
    nTimer = SetTimer(hWnd, 1, 50, (TIMERPROC)NULL); /* every 1/20 second */
    if (!nTimer){
        LoadString(hInst, S_TIMER, errBuf, SIZ_ERR_BUF);
        MessageBox(hWnd, errBuf, appName, MB_ICONSTOP | MB_APPLMODAL);
        PostMessage(hWnd, WM_CLOSE, 0, 0L);
    }
    initGraph(hWnd); /* init the graphics objects */
    break;

case WM_RBUTTONDOWN:
    if (appState == STAT_READ){
        if ((vtdStat & VS_LISTEN) && (!advFlg)){
            caution();
            advWord(hWnd);
        }
    }
    break;

case WM_LBUTTONDOWN:
    if (appState == STAT_READ){
        if ((vtdStat & VS_LISTEN) && (!advFlg))
            selWord(hWnd, LOWORD(lParam), HIWORD(lParam));
    }
    if (appState == STAT_PAGE){
        /*
        * not previous -
        * test if a request to goto the next page
        */
        if (!tstNext(hWnd, LOWORD(lParam), HIWORD(lParam))){
            /*
            * next page not requested -
            * test if a request to goto the previous page

```

```

        */
        if (curPage){
            tstPrev(hWnd, LOWORD(lParm), HIWORD(lParm));
        }
    }
}
break;

case WM_PAINT:
    BeginPaint(hWnd, &ps);

    if ((ps.rcPaint.left != ps.rcPaint.right) &&
        (ps.rcPaint.top != ps.rcPaint.bottom))
    {
#ifdef SKIP
        if ((ps.rcPaint.left == expLeft) &&
            (ps.rcPaint.right == expRight) &&
            (ps.rcPaint.top == expTop) &&
            (ps.rcPaint.bottom == expBot))
        {
#endif
            paint(hWnd);
            /*
             * if this is the first time the window has been drawn
             * load the speech recognition software
             */
            if (!initFlg){
                initFlg = TRUE;
                /*
                 * get the user ID
                 */
                rc = DialogBox(hInst, "ID", hWnd, &getID);
                if (lrc){
                    /*
                     * error opening user data file
                     */
                    PostMessage(hWnd, WM_CLOSE, 0, 0L);
                }else{
                    /*
                     * file created - log header info
                     */
                    wrtLog(hWnd, kidImage[curKid]);
                    wsprintf(cBuf, " age %d\n", age);
                    wrtLog(hWnd, cBuf);

                    wsprintf(cBuf, "Minimum score is %d\n", minScore);

```

```

        wrtLog(hWnd, cBuf);
        wsprintf(cBuf, "Rejection threshold is %d\n", rejectVal);
        wrtLog(hWnd, cBuf);

        /*
         * display the hour glass pointer
         */
        hCursor = LoadCursor(NULL, IDC_WAIT);
        SetCursor(hCursor);
        rc = vtdOpen(hWnd);
        if (rc != SM_RC_OK){
            PostMessage(hWnd, WM_CLOSE, 0, 0L);
            vtdStat = VS_ERROR;
        }
    }
}

#ifdef SKIP
    }else{
        expLeft = ps.rcPaint.left;
        expRight = ps.rcPaint.right;
        expTop = ps.rcPaint.top;
        expBot = ps.rcPaint.bottom;
        InvalidateRect( hWnd, NULL, TRUE);
    }
#endif
}
EndPaint(hWnd,&ps);
break;

case WM_DESTROY:
    /*
     * Disconnect from the Recognition Engine.
     */
    if (vtdStat & VS_CONN){
        SmDisconnect( 0, NULL, &reply );

        SmGetRc( reply, &rc);
        if (rc != SM_RC_OK){
            LoadString(hInst, S_VTD_DIS, msgBuf, SIZ_ERR_BUF);
            wsprintf(errBuf, "%s, rc= %d\n", msgBuf, rc);
            MessageBox(hWnd, errBuf, appName, MB_OK | MB_ICONHAND |
MB_APPLMODAL);
        }
    }

    if (vtdStat & VS_OPEN){

```

```

rc = SmClose();

if (rc != SM_RC_OK){
    LoadString(hInst, S_VTD_CLOSE, msgBuf, SIZ_ERR_BUF);
    wsprintf(errBuf, "%s, rc= %d\n", msgBuf, rc);
    MessageBox(hWnd, errBuf, appName, MB_OK | MB_ICONHAND |
MB_APPLMODAL);
}
}

/*
 * close the data file if open
 */
if (hFile != INVALID_HANDLE_VALUE){
    if (!CloseHandle(hFile)){
        errCode = GetLastError();
        fileErr(S_FILE_CLOSE, errCode);
    }
}

/*
 * if mci device active - close it
 */
if (mciActive)
mciSendCommand(uDeviceID, MCI_CLOSE, (DWORD)0, (DWORD)NULL);

/*
 * create a stock pen and brush to free custom objects
 */
hTempBrush = GetStockObject(BLACK_BRUSH);

/*
 * delete the brushes
 */
SelectObject(hDC, hTempBrush);

/*
 * delete the fonts
 */
SelectObject(hDC, hOldFont);
DeleteObject(hTextFont);
DeleteObject(hHiFont);

/*

```

```

        * delete the DCs and clear maps
        */
        DeleteDC(hPicDC);
        DeleteObject(hClrPicMap);

        /*
        * delete the off screen dc and bitmap
        */
        DeleteDC(hOffDC);
        DeleteObject(hOffMap);

        /*
        * delete the stretch bitmap and dc
        */
        if (!vgaFlg){
            DeleteDC(hStretchDC);
            DeleteObject(hStretchMap);
        }

        PostQuitMessage(0);
        break;

    case WM_SETCURSOR:
        SetCursor(hCursor);
        break;

    case MM_MCINOTIFY:
        mciSendCommand(uDeviceID, MCI_CLOSE, (DWORD)0, (DWORD)NULL);
        mciActive = FALSE;
        if (sndIns == sndRem){
            sndStat = SNQ_IDLE;
        }
        /*
        * sounds currently idle -
        * test for states needing the wave file to complete
        */
        switch (appState)
        {
        case STAT_CORRECT:
            /*
            * turn on the microphone
            */
            micOn(hWnd);
            advFin(hWnd); /* complete the advance */
            break;

        case STAT_MIC:

```

```

/*
 * turn on the microphone
 */
micOn(hWnd);
break;

case STAT_PERF:
if ((curIndex < wordCnt))
playNext(hWnd);
break;

case STAT_TOP:
perform(hWnd);
break;

        case STAT_WAIT:
            appState = STAT_PAGE;
            hCursor = LoadCursor(NULL, IDC_ARROW);
            SetCursor(hCursor);
            break;
    }
} else
sndOut(hWnd);
break;

#ifdef SKIP
    case WM_QUERYNEWPALETTE:
        RealizePalette(hDC);
        InvalidateRect(hWnd, NULL, TRUE);
        break;
#endif
default:
    return(DefWindowProc(hWnd, wMessage, wParam, lParam));
}
return(0L);
}

```

Here is the file TEXT.C

```
#include "read.h"
#include "struct.h"
#include "global.h"

/*
 * common storage for this module only
 */
static WORD  spcLeft;    /* space left on a line */
static WORD  yAdr;      /* y adr of current line */
static WORD  xAdr;      /* x adr of current word */
static WORD  charCnt;    /* count of characters in a word */
static char  *pWord;     /* ptr to current word */
static WORD  state;      /* parse state */
static int   margin;     /* left margin of a page */
static layOut *pForm;    /* ptr to page format */

void  initText(HWND hWnd)
{
    /*
     * local storage
     */
    char  *pStr;          /* ptr to string to be read */
    SIZE  extent;         /* text extents */
    BOOL  bldFlg;         /* continue building words */
    int   wordW;          /* width of word in pixels */
    int   i;              /* temp counter */

    /*
     * routines called
     */
    void addWord();        /* add a word to the list */

    /*
     * external storage referenced
     */
    extern char  book[];   /* the book's text */
    extern int   pageType[]; /* page format type table */

    /*
     * init the page index array
     */
}
```

```

for (i=0; i < MAX_PAGE; i++)
    pageIndex[i] = -1;

```

```

SelectObject(hDC, hHiFont);
numPages = 0;
pForm = pPageForms[pageType[numPages]];
pStr = book;
yAdr = pForm->y;
margin = pForm->left;
xAdr = margin;
pWord = pStr;
spcLeft = pForm->width;
wordCnt = charCnt = 0;
bldFlg = TRUE;
state = 0;
do {
    switch (*pStr)
    {
        case 0:
            /*
             * end of text
             */
            if (charCnt){

                /*
                 * if there was a word prior to the end of text
                 * add it to the word list
                 */
                addWord();
            }
            bldFlg = FALSE;
            break;

        case ' ':
            /*
             * space
             */
            if (state){
                /*
                 * if there was a word prior to the space
                 * add it to the word list
                 */
                addWord();
            }
            state = 0; /* reset state */
    }
}

```

```

/*
 * adjust the x adr for the space
 */
if (spcLeft != pForm->width){
/*
 * if not on the left margin of the line
 * add in the width for the space
 */
GetTextExtentPoint(hDC, pWord, 1, &extent);
wordW = (WORD) extent.cx;
if (wordW > spcLeft){
    yAdr += pForm->yInc;
    spcLeft = pForm->width;
    xAdr = margin;
}else{
    xAdr += wordW;
    spcLeft -= wordW;
}
}
pStr++;
pWord = pStr;
break;

case '"':
case '.':
case ',':
case '!':
case '?':
/*
 * if starting a word -
 * add the quote as a word
 */
if (!state && *pStr == '"'){
    state = ST_BEG;
    charCnt++;
    addWord();
    pStr++;
    pWord = pStr;

}else if (state == ST_WORD){
/*
 * if in a word -
 * terminate the word and add it to the list
 */
    addWord();

```

```

    /*
    * start parsing the ending punctuation
    */
    state = ST_END;
    charCnt++;
    pWord = pStr;
    pStr++;
} else {
    /*
    * if in ending punctuation -
    * parse the character
    */
    charCnt++;
    pStr++;
}
break;

case '#':
    /*
    * new paragraph
    */
    if (state) {
        /*
        * if there was a word prior to the space
        * add it to the word list
        */
        addWord();
    }
    /*
    * start a new line
    */
    state = 0; /* reset state */
    yAdr += (pForm->yInc * 3) / 2;
    spcLeft = pForm->width;
    xAdr = margin;
    pStr++;
    pWord = pStr;
    break;

case '@':
    /*
    * new page
    */
    if (state) {
        /*
        * if there was a word prior to the space

```

```

        * add it to the word list
        */
        addWord();
    }
    /*
    * start a new page
    */
    numPages++;

    state = 0; /* reset state */
    pForm = pPageForms[pageType[numPages]];
    yAdr = pForm->y;
    margin = pForm->left;
    xAdr = margin;
    pWord = pStr;
    spcLeft = pForm->width;
    pStr++;
    pWord = pStr;
    break;

case '*':
    /*
    * new column
    */
    if (state){
        /*
        * if there was a word prior to the space
        * add it to the word list
        */
        addWord();
    }
    /*
    * start a new column
    */
    state = 0; /* reset state */
    yAdr = pForm->y;
    margin = pForm->right;
    xAdr = margin;
    pWord = pStr;
    spcLeft = pForm->width;
    pStr++;
    pWord = pStr;
    break;

default:
    /*

```

```

        * normal character - parse it for now
        */
        state = ST_WORD;
        charCnt++;
        pStr++;
        break;
    }
} while (bldFlg);
wordFlg = FALSE; /* flag the words have been built */
curIndex = -1; /* flag there is no highlighted word */
}

/*
 * add a word to the list
 */
void addWord()
{
    /*
     * local storage
     */
    SIZE extent; /* text extents */
    int wordW; /* width of word in pixels */
    WORD prev; /* prev word index */

    GetTextExtentPoint(hDC, pWord, charCnt, &extent);
    wordW = (WORD) extent.cx;
    if (wordW > spcLeft){
        /*
         * there is no space left on the line
         * begin a new line
         */
        yAdr += pForm->yInc;
        spcLeft = pForm->width;
        xAdr = margin;

        /*
         * need to find if there is beginning punctuation
         */
        if (state == ST_END)
            prev = wordCnt - 2;
        else if (state == ST_WORD)
            prev = wordCnt - 1;
        else
            prev = (WORD) -1;
        if ((prev > 0) && (words[prev].type == ST_BEG)){

```

```

        words[prev].x = xAdr;
        words[prev].y = yAdr;
        xAdr += words[prev].width;
        spcLeft -= words[prev].width;
    }

    /*
     * if in ending punctuation move the preceeding word
     */
    if (state == ST_END){
        words[wordCnt-1].x = xAdr;
        words[wordCnt-1].y = yAdr;
        xAdr += words[wordCnt-1].width;
        spcLeft -= words[wordCnt-1].width;
    }
}

words[wordCnt].x = xAdr;
words[wordCnt].width = wordW;
words[wordCnt].y = yAdr;
words[wordCnt].str = pWord;
words[wordCnt].cnt = charCnt;
words[wordCnt].type = state;
words[wordCnt].sndType = PLAY_NONE;
words[wordCnt].page = numPages;
if (pageIndex[numPages] < 0)
    pageIndex[numPages] = wordCnt;
wordCnt++;
charCnt = 0;
xAdr += wordW;
spcLeft -= wordW;
}

/*
 * display the text
 */

void dspText(HWND hWnd)
{
    /*
     * local storage
     */
    int i;          /* temp counter */
#ifdef SIZEIT
    RECT rect;
#endif
}

```

```

int index; /* display active words index */

/*
 * routines called
 */
void nextWord(HWND); /* display the next word to read */
BOOL wrtLog(HWND, char *); /* write to the log file */
void hiLight(int); /* highlight a word of text */

/*
 * if not ready to listen - do not display the text
 */
if (!(vtdStat & VS_LISTEN)){
    txtFlg = TRUE; /* flag ready to display text */
    return;
}
SelectObject(hDC, hTextFont);
SetTextColor(hDC, textColor);
SetBkMode(hDC, TRANSPARENT);

for (i = pageIndex[curPage]; (i < wordCnt) && (words[i].page == curPage); i++) {
#ifdef SIZEIT
    if (words[i].type == ST_WORD){
        rect.left = words[i].x;
        rect.right = words[i].x + words[i].width;
        rect.top = words[i].y - pLay->top;
        rect.bottom = words[i].y + pLay->bot;
        FillRect(hDC, &rect, GetStockObject(GRAY_BRUSH));
    }
#endif
    TextOut(hDC, words[i].x, words[i].y, words[i].str, words[i].cnt);
}
if (curIndex >= 0){
    /*
     * highlight the words being played
     */
    index = begIndex;
    while (index <= curIndex){
        if (((appState == STAT_READ) && (words[index].type == ST_WORD)) ||
            (words[index].sndType != PLAY_NONE))
        {
            hiLight(index);
        }
        index++;
    }
}

```

```

}else{
    if (appState == STAT_READ){
        nextWord(hWnd); /* first time on this page */
        wsprintf(cBuf, "Next Story Word - %s \n", curWord);
        wrtLog(hWnd, cBuf);
    }
}
}

void nextWord(HWND hWnd)
{
    /*
    * routines called
    */
    void copyWord(short, char *); /* copy a story word into curWord */
    void sndAdd(HWND, int, int); /* play a sound wave file */
    void normal(int); /* rewrite highlight text as normal */
    void hiLight(int); /* highlight a word of text */
    BOOL micOff(HWND); /* turn off the microphone */

    /*
    * display the previous word normally again
    */
    SetBkMode(hDC, TRANSPARENT);
    if (curIndex >= 0){
        normal(curIndex);
    }

    /*
    * pt to the next word in the story
    */
    do{
        curIndex++;
        if ((curIndex >= wordCnt) || (words[curIndex].page > curPage)){
            curIndex = -1;
            curWord[0] = 0;
            appState = STAT_WAIT;
            hCursor = LoadCursor(NULL, IDC_WAIT);
            SetCursor(hCursor);
            micOff(hWnd); /* turn off the microphone */
            machMic = FALSE;
            sndAdd(hWnd, SND_PHR, PHR_END_READ); /* cheer at the end of the reading */
            return;
        }
    }while (words[curIndex].type != ST_WORD);
}

```

```

copyWord(curIndex, curWord); /* copy the story word into curWord */

/*
 * highlight the current word
 */
hiLight(curIndex);
trys = 0; /* reset the trys counter */
begIndex = curIndex;
}

/*
 * advWord - user requested advance to the next word
 */
void advWord(HWND hWnd)
{
/*
 * local storage
 */
BOOLstatus; /* microphone status */

/*
 * routines called
 */
BOOL micOff(HWND); /* turn off the microphone */
void advFin(HWND); /* advance to the next word */
void playCorrect(HWND); /* play the correction phrases */

if (feedBack){
    advFlg = TRUE; /* flag now advancing to the next word */
    status = micOff(hWnd); /* turn off the microphone */
machMic = FALSE;
    if (status) /* mic already off - play the correction */
        playCorrect(hWnd);
else
    appState = STAT_REQCOR;

    }else{
        advFin(hWnd);
    }
}

/*
 * playCorrect - play the correct word
 */
void playCorrect(HWND hWnd)

```

```

{
    /*
     * routines called
     */
    void sndAdd(HWND, int, int); /* play a sound wave file */

    appState = STAT_CORRECT;
    sndAdd(hWnd, SND_PHR, corWav); /* play the correction wave file */
    /*
     * advance to the next wave file
     */
    corWav++;
    if (corWav >= PHR_COREND)
        corWav = PHR_CORBEG;

    sndAdd(hWnd, SND_WORD, curIndex); /* play the current word */
}

/*
 * advFin - complete the user request to advance to the next word
 */
void advFin(HWND hWnd)
{
    /*
     * routines called
     */
    BOOL wrtLog(HWND, char *); /* write to the log file */
    void nextWord(HWND); /* display the next word to read */

    wsprintf(cBuf, "Advanced past - %s\n\n", curWord);
    wrtLog(hWnd, cBuf);

    /*
     * set playback entry to be play a wave file
     */
    words[curIndex].id = curIndex;
    words[curIndex].sndType = PLAY_WAVE;

    nextWord(hWnd); /* pt to the next word to be read */

    /*
     * log the next word to be recognized
     */
    wsprintf(cBuf, "Next Story Word - %s\n", curWord);
}

```

```

wrtLog(hWnd, cBuf);

advFlg = FALSE; /* flag now complete */
}

/*
 * selword - select the word currently under the mouse
 */
void selWord(HWND hWnd, WORD xPos, WORD yPos)
{
    /*
     * local storage
     */
    int i; /* temp counter */

    /*
     * routines called
     */
    void normal(int); /* rewrite highlight text as normal */
    void hiLight(int); /* highlight a word of text */
    void copyWord(short, char *); /* copy a story word into curWord */
    BOOL tstNext(HWND, WORD, WORD); /* test if next page button hit */
    BOOL tstPrev(HWND, WORD, WORD); /* test if previous page button hit */
    BOOL tstMic(HWND, WORD, WORD); /* test if microphone button hit */
    BOOL wrtLog(HWND, char *); /* write to the log file */

    /*
     * test if a request to go to the previous page
     */
    if (curPage){
        if (tstPrev(hWnd, xPos, yPos))
            return;
    }

    /*
     * test if a request to go change mic state
     */
    if (dspMic){
        if (tstMic(hWnd, xPos, yPos))
            return;
    }

    /*
     * test if a request to goto the next page
     */

```

```

if (IsNext(hWnd, xPos, yPos))
    return;

```

```

/*
 * look for a word that is under the cursor
 */
for (i=pageIndex[curPage]; (i < wordCnt) && (words[i].page == curPage); i++){

    if (((words[i].y - pLay->top) <= yPos) &&
        ((words[i].y + pLay->bot) >= yPos) &&
        (words[i].x <= xPos) &&
        ((words[i].x + words[i].width) >= xPos))
    {
        if (words[i].type == ST_WORD){
            /*
             * clicked on a valid word - advance to it
             * display the previous word normally again
             */
            SetBkMode(hDC, TRANSPARENT);
            if (curIndex >= 0){
                normal(curIndex);
            }
            /*
             * move the word into the current word buffer
             */
            begIndex = curIndex = i;
            copyWord(curIndex, curWord); /* copy the story word into curWord */

            /*
             * highlight the current word
             */
            hiLight(curIndex);
            trys = 0; /* reset the trys counter */

            /*
             * log that the new word was selected
             */
            sprintf(cBuf, "\nSelected - %s\n", curWord);
            wrtLog(hWnd, cBuf);
        }
        break;
    }
}
}

```

```

void copyWord(short index, char *pBuf)
{
    /*
     * local storage
     */
    char *pSrc; /* temp ptr to word */
    short i; /* temp counter */

    /*
     * move the word into the current word buffer
     */
    pSrc = words[index].str;
    for (i=0; i < words[index].cnt; i++)
        *pBuf++ = *pSrc++;
    *pBuf = 0;
}

/*
 * hiLight - highlight a word
 */
void hiLight(int index)
{
    /*
     * local storage
     */
    RECT rect;

    SetTextColor(hDC, hiTextColor);
    SelectObject(hDC, hHiFont);
    rect.left = words[index].x;
    rect.right = words[index].x + words[index].width;
    rect.top = words[index].y - pLay->top;
    rect.bottom = words[index].y + pLay->bot;
    FillRect(hDC, &rect, GetStockObject(WHITE_BRUSH));
    TextOut(hDC, words[index].x, words[index].y-pLay->hiAdj,
            words[index].str, words[index].cnt);
}

/*
 * normal - display highlighted text normally
 */
void normal(int index)
{
    /*
     * local storage
     */

```

```
RECT rect;
```

```
SelectObject(hDC, hTextFont);
SetTextColor(hDC, textColor);
rect.left = words[index].x-2;
rect.right = words[index].x + words[index].width;
rect.top = words[index].y - pLay->top;
rect.bottom = words[index].y + pLay->bot;
FillRect(hDC, &rect, GetStockObject(WHITE_BRUSH));
TextOut(hDC, words[index].x, words[index].y,
        words[index].str, words[index].cnt);
```

```
}
```

```
/*
```

```
* caution - caution light a word
```

```
*/
```

```
void caution()
```

```
{
```

```
    SetTextColor(hDC, wrTextColor);
    SelectObject(hDC, hHiFont);
    TextOut(hDC, words[curIndex].x, words[curIndex].y-pLay->hiAdj,
            words[curIndex].str, words[curIndex].cnt);
```

```
}
```

```
/*
```

```
* tstPrev - test if request to go to previous page
```

```
*/
```

```
BOOL tstPrev(HWND hWnd, WORD xPos, WORD yPos)
```

```
{
```

```
/*
```

```
* routines called
```

```
*/
```

```
void normal(int); /* rewrite highlight text as normal */
void hiLight(int); /* highlight a word of text */
void copyWord(short, char *); /* copy a story word into curWord */
void chgVocab(HWND); /* change vocabulary */
void paint(HWND); /* repaint the window */
```

```
/*
```

```
* external storage referenced
```

```
*/
```

```
extern int pageType[]; /* page format type table */
```

```

/*
 * test if cursor is over the button
 */
if (((pButton->prev.top) <= yPos) &&
    ((pButton->prev.bottom) >= yPos) &&
    (pButton->prev.left <= xPos) &&
    (pButton->prev.right >= xPos))
{
    /*
     * previous button has been hit
     */
    /* back up 1 page */
    curPage--;
    curBack = curPage + BG_READ;
    pLay = pPageForms[pageType[curPage]];

    curIndex = pageIndex[curPage];
    while ( words[curIndex].type != ST_WORD)
        curIndex++;
    begIndex = curIndex;

    /*
     * display the new background
     */
    appState = STAT_READ;
    paint(hWnd);

    /*
     * switch to this page's vocabulary
     */
    chgVocab(hWnd);

    /*
     * move the word into the current word buffer
     */
    copyWord(curIndex, curWord); /* copy the story word into curWord */

    trys = 0; /* reset the trys counter */

    /*
     * log that the new page was selected
     */
    wprintf(cBuf, "\nPrevious Page Requested - %s\n", curWord);
}

```

```

    wrtLog(hWnd, cBuf);
    wsprintf(cBuf, "Next Story Word - %s\n", curWord);
    wrtLog(hWnd, cBuf);
    return(TRUE);
}
return(FALSE);
}

/*
 * tstNext - test if request to go to next page
 */
BOOL tstNext(HWND hWnd, WORD xPos, WORD yPos)
{
    /*
     * routines called
     */
    void normal(int); /* rewrite highlight text as normal */
    void hiLight(int); /* highlight a word of text */
    void copyWord(short, char *); /* copy a story word into curWord */
    void chgVocab(HWND); /* change vocabulary */
    void begPerf(HWND); /* start the performance */
    void paint(HWND); /* repaint the window */

    /*
     * external storage referenced
     */
    extern int pageType[]; /* page format type table */

    /*
     * test if cursor is over the button
     */
    if (((pButton->next.top) <= yPos) &&
        ((pButton->next.bottom) >= yPos) &&
        (pButton->next.left <= xPos) &&
        (pButton->next.right >= xPos))
    {
        /*
         * next button has been hit
         */
        curPage++;
        /*
         * at the end of the book - start the performance
         */
        if (curPage >= numPages){
            begPerf(hWnd);
        }
    }
}

```

```

}else{
    /*
    * setup to begin reading on the next page
    */
    curBack = curPage + BG_READ;
    pLay = pPageForms[pageType[curPage]];
    curIndex = pageIndex[curPage];
    while ( words[curIndex].type != ST_WORD)
        curIndex++;
    begIndex = curIndex;

    /*
    * display the new background
    */
    appState = STAT_READ;
    paint(hWnd);

    /*
    * switch to this page's vocabulary
    */
    chgVocab(hWnd);

    /*
    * move the word into the current word buffer
    */
    copyWord(curIndex, curWord); /* copy the story word into curWord */

    trys = 0; /* reset the trys counter */

    /*
    * log that the new word was selected
    */
    wsprintf(cBuf, "\nNext Page Requested - %s\n", curWord);
    wrtLog(hWnd, cBuf);

    wsprintf(cBuf, "Next Story Word - %s \n", curWord);
    wrtLog(hWnd, cBuf);
}
return(TRUE);
}
return(FALSE);
}

/*
* tstMic - test if request to change mic state

```

```
*/
BOOL tstMic(HWND hWnd, WORD xPos, WORD yPos)
{
    /*
    * routines called
    */
    void dspBack(HWND,WORD,int,int); /* display the background frame */
    int micOn(HWND); /* turn on the microphone */
    BOOL micOff(HWND); /* turn off the microphone */

    /*
    * test if cursor is over the button
    */
    if (((pLay->mic.y) <= yPos) &&
        ((pLay->mic.y + pBack[BG_MIC_ON].height) >= yPos) &&
        (pLay->mic.x <= xPos) &&
        ((pLay->mic.x + pBack[BG_MIC_ON].width) >= xPos))
    {
        /*
        * mic button has been hit
        */
        if (userMic){
            /*
            * currently on - turn it off
            */
            userMic = FALSE;
            micOff(hWnd);
            dspBack(hWnd, BG_MIC_OFF, pLay->mic.x, pLay->mic.y);
        }else{
            /*
            * currently off - update its state
            */
            userMic = TRUE;
            if (machMic)
                micOn(hWnd);
            else
                dspBack(hWnd, BG_MIC_BUSY, pLay->mic.x, pLay->mic.y);
        }
        return(TRUE);
    }
    return(FALSE);
}
```